

# On multiply-exponential write-once Turing machines

Maciej Zielenkiewicz<sup>a,\*</sup>, Aleksy Schubert<sup>a,\*</sup>, Jacek Chrząszcz<sup>a,\*</sup>

<sup>a</sup>*Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland*

---

## Abstract

In this work we analyze the multiply-exponential complexity classes for write-once Turing machines, i.e. machines that can write to a given tape cell at most once. We show that  $k\text{-DEXPWOSPACE} = k\text{-DEXPWOTIME} = k\text{-EXPTIME}$  and the nondeterministic counterpart. For alternating machines we show that  $k\text{-AEXPWOTIME} = k\text{-AEXPTIME} = k\text{-1-EXPSPACE}$ .

*Keywords:* computational complexity, write-once Turing machine

---

## 1. Introduction

The idea of write-once machines was first studied by Hao Wang [1] in 1957. Those machines are analogous to the Turing machines, with the exception that writing is possible only to blank spaces; this model fits many types of storage which are used both today and historically, for example punch cards and tapes and recordable optical discs (CD-R, DVD-R etc.). Those machines are equivalent to Turing machines with respect to the languages they accept. In this paper we characterize some complexity classes of write-once machines and show what are the corresponding complexity classes for the usual Turing machines. Additionally, we consider a further restriction of write-once machines, that are allowed to write only at the end of the tape, and show that this class of machines is not universal.

## 2. Known results

In 1960 Lee [2] has shown many useful conversions of programs for usual Turing machines to programs for write-only machines with their complexities. Subsequently Rivest and Shamir [3] have shown a coding scheme which allows for efficient simulation of updates; one of their most important results is that a value can be stored in a way which permits  $t$  updates at the expense of increasing the storage size asymptotically  $t/\log t$  times. We will be using this

---

<sup>☆</sup>This work is supported by Polish NCN grant DEC-2012/07/B/ST6/01532.

<sup>\*</sup>Corresponding authors.

*Email addresses:* [maciej.zielenkiewicz@mimuw.edu.pl](mailto:maciej.zielenkiewicz@mimuw.edu.pl) (Maciej Zielenkiewicz), [alx@mimuw.edu.pl](mailto:alx@mimuw.edu.pl) (Aleksy Schubert), [chrzaszcz@mimuw.edu.pl](mailto:chrzaszcz@mimuw.edu.pl) (Jacek Chrząszcz)

result extensively through the paper. The coding schemes for efficient updates are still an active area of research (for example see [4]), with new applications in efficient usage of flash memory, as the erase operation is much more expensive (i.e. is slower and causes degradation) than a simple write.

An analysis of complexity of some algorithms was done by Irani et al ([5]), who show that an algorithm with a given complexity using  $k$  space in the usual computing model can be rewritten to the write-only (deterministic) RAM machine model (with a constant amount of additional multiple-write memory) with an increase of running time by a factor of  $\log n / \log \log n$  or by a factor of  $\log k$ . It is also shown that in general there is no simulation technique which uses less write-only space than the running time of the original algorithm. There are some important data structures which can be simulated as fast as the original algorithm, which include binary search trees and find-union data structures. Eventually in their paper it is shown that  $\text{WO-PSpace} = \text{P}$ , where WO denotes that the complexity class when using write-once memory.

A slightly different model was presented by Vitter in [6], where effects connected with usage of optical disks were taken into account. The model assumed a division of memory into blocks, where each write of a block would come with a need for additional memory for the block header, therefore efficient ways of aggregating writes need to be used. Their main result is that for block size  $B$  and  $n$  allocated regions on a disc the time needed for an I/O operation is  $\log n + B \log B$ .

### 3. Basic definitions

For simplicity we assume that the alphabet of the Turing machine is always just two symbols, and we call a cell blank (0) or marked (1). We can easily simulate a bigger alphabet by encoding a symbol in more than one cell. In defining the write-once Turing machine we follow definitions set forth in the paper [1] by Hao Wang, with a slight change that we will assume that the tape is infinite only in one direction. Again, the reduction to the machine with two-way infinite tape is not complicated, although it would have complicated the proofs unnecessarily — we can pick a constant  $C$  and have the machine mark every  $C$ th cell; we will call such a cell a length-marking cell. Then the machine can find the beginning of the tape by going to the last cell it has marked as length-marking, and repeat going  $C$  cells to the left and checking if the cell is marked until we find an unmarked cell, which is the cell directly preceding the beginning cell of the tape. The definition of a write-once Turing machine is mostly the usual definition of the Turing machine, i.e. it is a machine with one tape and one read/write head and a two symbol alphabet, but it is not allowed to write a blank to a cell which contains a non-blank symbol. It is not an error to try to mark again the cell which is already marked, although it does not change the state of the tape.

The use of the Turing machine model is in contrast to the RAM machine model used in the analysis of WO-PSpace by Irani et al. ([5]). We show how to prove their result in our model in Theorem 1. Our model naturally extends

to non-deterministic and alternating versions, and we will be using letters “D” for deterministic, “N” for non-deterministic and “A” for alternating versions.

#### 4. Our reductions

We show how some of the complexity classes for write-once machines relate to those for standard Turing machines. More specifically we prove that  $k\text{-EXPTIME} = k\text{-DEXPWOSPACE} = k\text{-DEXPWOTIME}$ . We also prove that the same equations hold if we replace all the classes with their nondeterministic counterparts. Moreover, we show how  $k\text{-EXPSPACE}$  relates to alternating write-once classes by showing that  $k\text{-AEXPWOTIME} = k\text{-AEXPTIME} = k\text{-EXPSPACE}$ .

**Theorem 1.**  $\text{DWO-PSpace} = \text{P}$ .

PROOF. See proof of theorem 2 below, substituting 0 for  $k$ .

**Theorem 2.**  $k\text{-EXPTIME} = k\text{-DEXPWOSPACE}$ .

PROOF. First we show that  $k\text{-EXPTIME} \subseteq k\text{-DEXPWOSPACE}$ . Suppose we have a  $k\text{-EXPTIME}$  machine which has running time on input of length  $n$

bounded by the function  $f(n) = 2^{\overbrace{\dots}^{k+2} n^t}$ . Then it can write a tape of length at most  $f(n)$  updating each location at most  $f(n)$  times. To store a representation of a tape of this size permitting the expected number of updates we

will need at most  $f(n) \cdot f(n) = 2^{\overbrace{\dots}^{k+2} n^t} \cdot 2^{\overbrace{\dots}^{k+2} n^t} = 2^{\overbrace{\dots}^{k+2} n^{t+1}}$  memory, which is in  $k\text{-DEXPWOSPACE}$ .

The other inequality is  $k\text{-EXPTIME} \supseteq k\text{-DEXPWOSPACE}$ , which is more tricky. Let  $\mathcal{A}$  be our  $k\text{-DEXPWOSPACE}$  machine which makes at most  $g(n)$  writes. We need to show how many steps are needed to simulate the execution between writes. We show a lemma for that:

**Lemma 3.** *A  $k\text{-DEXPWOSPACE}$  machine  $\mathcal{A}$  with space complexity  $f(n) = 2^{\overbrace{\dots}^{k+2} n^t}$  started in a configuration in which it eventually stops makes at most  $f(n) \cdot s$  steps between any two consecutive writes.*

PROOF. Suppose that  $\mathcal{A}$  has  $s$  states. Between two consecutive writes we can treat the machine as a two-way DFS automaton (the tape does not change). At most  $f(n)$  of tape is used at any time. The automaton may visit a location on the tape at most  $s$  times, because otherwise some location on tape is visited at least twice in the same state between two consecutive writes. The sequence of states

between the two visits of the same location in the same state could be repeated any number of times, therefore the automaton would not stop — contradiction with assumption. Therefore there are at most  $f(n) \cdot s$  steps between any two consecutive writes.

**Lemma 4.** *A  $k$ -NEXPWOSPACE machine  $\mathcal{A}$  with space complexity  $f(n) =$*

$$\underbrace{2^{2^{\dots^{2^{k+2}}}}}_{n^t}$$

*started in a configuration in which it eventually stops has a run that makes at most  $f(n) \cdot s$  steps between any two consecutive writes.*

PROOF. The proof is very similar to this in Lemma 3, but we show how to shorten the runs instead of showing nontermination. More specifically, we can simulate the operations of the non-deterministic two-way finite automaton between two consecutive writes. In case the automaton is twice in the same state in a given location on the tape we can just skip the computation steps between the two visits. This manipulation performed iteratively results in a run that visits each location in each state at most once. The number of steps is therefore  $f(n) \cdot s$ .

PROOF OF THEOREM 2 - CONTINUATION. Our simulation is in  $k$ -EXPTIME, as the time needed for simulation can be computed as

$$(\# \text{ of writes}) \cdot (\# \text{ of steps between two writes}) = g(n) \cdot O(g(n)) = O((g(n))^2).$$

**Theorem 5.**  $k$ -NEXPTIME =  $k$ -NEXPWOSPACE.

PROOF. The proof is the same as the proof of the Theorem 2, except that we have to use Lemma 4 instead of Lemma 3.

**Theorem 6.**  $k$ -DEXPWOTIME =  $k$ -EXPTIME.

PROOF. Notice that  $k$ -DEXPWOTIME  $\subseteq$   $k$ -EXPTIME is trivial, since the only difference between write-once machine and usual Turing machine is a *restriction* on the eligible transitions.

To prove  $k$ -DEXPWOTIME  $\supseteq$   $k$ -EXPTIME we need to show how to simulate a  $k$ -EXPTIME machine with a  $k$ -DEXPWOTIME one. Suppose that the  $k$ -

$$\underbrace{2^{2^{\dots^{2^{k+2}}}}}_{n^n}$$

EXPTIME machine has time complexity of  $f(n) = 2^{2^{\dots^{2^{k+2}}}}$ . The only difficulty with simulating the writes is the restriction of non-overwriting memory, but we can copy the contents of the entire tape on each write that changes a marked cell to blank. The cost of simulation of a single write is  $f(n)^2$ , and reads can be performed without any additional steps. Therefore the running time will be  $O((f(n))^3)$ , which does not drive us outside the  $k$ -EXPTIME complexity class.

**Theorem 7.**  $k$ -NEXPWOTIME =  $k$ -NEXPTIME and  $k$ -AEXPWOTIME =  $k$ -AEXPTIME (=  $(k-1)$ -EXPSpace).

PROOF. The above construction remains unchanged if we consider nondeterministic/alternating automata. In case of the alternating automata, the alternating states can be rewritten to use “fresh” memory.

#### 4.1. A note on automata writing at the end of tape.

A natural next step of our model simplification is to consider automata which write only at the end of the tape. We will show that such automata are not universal. Suppose we have an alphabet with a blank symbol and at least two non-blank symbols. We can introduce a restriction which does not allow blank symbol in the portion of the tape which was already written, i.e. no blanks between non-blank symbol. We will show how to simulate an automaton with this restriction while not keeping a full copy of its memory.

First we convert our machine so that it always goes from left to right and then to the beginning of the tape and so on. Then we show a pumping lemma operating on the automaton’s memory.

**Lemma 8.** *For a given WO-automaton  $\mathcal{A}$  with  $k$  states which writes only at the end of the tape, suppose  $s$  is a state in which  $\mathcal{A}$  writes to the tape. Then there exists an automaton  $\mathcal{A}_s$  which simulates the run of  $\mathcal{A}$  from  $s$  to the next writing state, which moves to the beginning of the tape and then only reads the tape from left to right.*

PROOF. When running from  $s$  to the next writing state,  $\mathcal{A}$  does no writes, so can be seen as a simple automaton with only input and not output (the symbol to be written can be stored in the state). Results of Vardi [7] show that such an automaton can be converted to one-way automaton at the expense of increasing the number of states to  $\exp k$ .

**Lemma 9 (Pumping lemma).** *For write-only automaton  $\mathcal{A}$  which writes only at the end and has  $k$  states, we can pump any word of length at least  $p(k) = 2^{\exp(k)^k}$  with respect to runs between writes with a single pumping scheme for all states.*

PROOF. We have at most  $k$  writing states. Using Lemma 8 we build automata for all of these states. Then we pump the product automaton of all those automata, which has  $K = (\exp(k^2))^k$  states, so words longer than  $O(2^K)$  can be pumped down.

We use our pumping lemma (Lemma 9) to shorten the runs between writes. The pumping length depends only on the number of states, so the maximum run length which we would need to store is limited by a constant; however we need some operations to do the pumping. We will run the automaton for  $2 \cdot p(n)$  steps from the beginning and for  $p(n)$  steps backwards from the end (this needs nondeterminism, but we can simulate it with deterministic automaton at an exponential cost in term of the number of states) and find the state which occurs at least twice (positions  $i_1, i_2$ ) in the normal run and at least once ( $i_3$ )

in the backwards run. Then we can conclude that the tape could have been pumped down between positions  $i_1$  and  $i_3$ , so we do not need to simulate the automaton any further in the reading run and we can continue from the writing state. If there is more than one writing state we simply have to have a bigger set of states in the beginning of the backwards simulation.

Then we do not need to store the whole tape contents of an automaton to simulate its run, as we can pump the tape down, and we know that the portion which was pumped down will not change. This means that a machine of this class can be simulated with an amount of memory which depends only on the number of states plus the size of the input, so the acceptance problem for this class of machines can be decided in PSPACE.

## 5. Conclusions

We have analyzed the multiply-exponential complexity classes for write-once Turing machines and shown basically that, as far as such big complexities are concerned, write-once Turing machines perform as good (or as bad) as the regular Turing machines. The only difference is that corresponding space and time classes are equal for write-once machines, whereas this remains unknown for regular Turing machines. If further restriction on write-once machines is applied, namely that writing is allowed only at the end of the tape, the resulting class of machines is not universal.

## References

- [1] H. Wang, A variant to Turing's theory of computing machines, J. ACM 4 (1) (1957) 63–92. doi:10.1145/320856.320867.
- [2] C. Y. Lee, Categorizing automata by W-machine programs, J. ACM 8 (3) (1961) 384–399. doi:10.1145/321075.321082.
- [3] R. L. Rivest, A. Shamir, How to reuse a “write-once” memory, Inf. and Control 55 (1-3) (1982) 1 – 19. doi:10.1016/S0019-9958(82)90344-8.
- [4] A. Shpilka, Capacity-achieving multiwrite WORM codes, IEEE Trans. Inf. Theory 60 (3) (2014) 1481–1487. doi:10.1109/TIT.2013.2294464.
- [5] S. Irani, M. Naor, R. Rubinfeld, On the time and space complexity of computation using write-once memory or is pen really much worse than pencil?, Math. syst. theory 25 (2) (1992) 141–159. doi:10.1007/BF02835833.
- [6] J. S. Vitter, An efficient I/O interface for optical disks, ACM Trans. Database Syst. 10 (2) (1985) 129–162. doi:10.1145/3857.3862.
- [7] M. Y. Vardi, A note on the reduction of two-way automata to one-way automata, Information Processing Letters 30 (5) (1989) 261 – 264. doi: [http://dx.doi.org/10.1016/0020-0190\(89\)90205-6](http://dx.doi.org/10.1016/0020-0190(89)90205-6).